Dynamic Trustiness Authentication Framework Based on Software's Behavior Integrity

Guojun PENG Xuanchen PAN Huanguo ZHANG Jianming FU
Computer School, Wuhan University, Wuhan 430072, Hubei, China
The Key Laboratory of Aerospace Information Security and Trust Computing, Ministry of Education, China
guojpeng@whu.edu.cn; tompanpan@163.com; liss@whu.edu.cn; jmfu@whu.edu.cn

Abstract

A dynamic trustiness authentication framework based on the integrity of software's behavior is proposed in this paper. The method to extract SIBDS (Software Intended Behaviors Describing Sets) and SBAC (Software Behavior Authentication Code) from the binary executable is introduced. In the framework, when the software begin to run, it should be monitored by SBMC (Software Behavior Monitoring Center), then the real API function invoking sequence will be acquired. The framework uses the software behavior comparison algorithm to verify whether the API invoking sequence gotten from the actual behavior is in accordance with SBAC; thereby the software's dynamic trustiness can be detected and guaranteed. The experiment results demonstrate the efficacy of the dynamic trustiness authentication framework.

Keywords: Function invoking, dynamic trustiness, behavior trustiness, behavior integrity

1. Introduction

Dynamic trustiness is indicated through comprehensive and deep understanding of the unconsistency between the intended software's behavior and the practical behavior. To ensure the dynamic trustiness is to make sure that the entity's behavior is always works in the intended way and goes towards the intended direction. The dynamic trustiness's related theory and technology, based on the software behavior, is a generally acknowledged fundamental problem with great difficulty and also a significant problem that must be solved.

Some domestic enterprises, university and related research institutes have done a lot study and research actively in the field of trusted computing [1-4]. Meanwhile, a series of practical work focused in software behavior intrusion detection has been done in the field of host intrusion detection. In 1996, Stephanie Forrest and his colleagues proposed a model for detecting intrusions at the level of privileged process using sequences of system calls: N-gram model, which is a typical representative of intrusion detection techniques based on processes' behavior [5]. Subsequently, inspired by Forrest, more outstanding models and achievements have spring up [6-11].

Currently, large quantities of research achievements have been accumulated in the field of software static analysis and intrusion detection both here and abroad. But in the field of trusted computing, the research on the software dynamic trustiness authentication model continues to be scarce.

In this paper, we put focus on the research of the software behavior and proposed a framework of dynamic software's trustiness authentication, based on the software behavior integrity.

2.Description of the Dynamic Trustiness Authentication Framework

2.1 General Framework

To introduce the framework we proposed, we make the following definitions.

Definition 1: Software Behavior. Software Behavior are any changes, influences or any operations made to the other independent entities when the software works as an independent entity.

Definition 2: Software Behavior Integrity. Software Behavior Integrity means the consistency between the intended behavior of the software and the real behavior in the real executing process of the software.

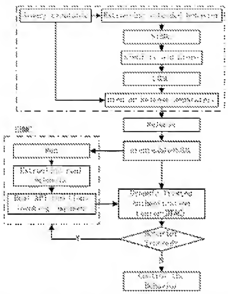


Figure 1. Dynamic Trustiness Authentication Model Based on Software's Behavior Integrity

In this paper, we consider that dynamic trustiness means the software works in the intended way, say the software behavior integrity is not broken. Software has dynamic state in the process of execution, so that the software is trusted at some time does not means it is trusted in the process of execution all the time. For the sake of this, the authentication procedure is sustaining and accompanying with the real executing process of the program.

To make the dynamic trustiness authentication more effectively, we propose the dynamic trustiness authentication framework on the basis of the software integrity, as is shown in Figure 1.

Firstly, we do the static analysis on the program's binary code to get the executing flow graph so that we can extract the software's intended behavior describing set (we describe the behavior using API functions invoking in this paper) from the software's intended behavior. Then, we simplify the software intended behavior describing set and store the set in the way we defined and get the SBAC (*Software Behavior Authentication Code*), which can be released along with the software or can be released by some authoritative websites.

When the software is in real executing process, SBMC (Software Behavior Monitoring Center) monitors the real executing process and get the trace of the software's execution (we describe it using API functions invoking sequence in this paper). Then, DTAC (Dynamic Trustiness Authentication Center) will match the real API function invoking sequence with the SBAC to check the software behavior integrity. If the behavior integrity is not broken, the current state of the software is trusted, or the current state is not trusted and SBCC (Software Behavior Control Center) must intervene and take corresponding measure to control the unknown behavior.

2.2 Theory of the Software Behavior's Trustiness Authentication

The software state k is influenced by the module behavior M, and whether or not the trusted state k can conduct the trust transitive depends on trustiness of the module behavior M that drives the state transition. In the proving process of the software behavior (process) trustiness, the intended behavior is represented by the trustiness behavior intended policies of the interpellant (DTAC).

Any process module that is inconsistent with the intended behavior policy will be regarded as unauthentic behavior, say the corresponding software process state is regarded as trustless, so that the discrimination process of the software behavior's trustiness can be converted into the trustiness verification of the software module' behavior.

One time behavior of a specific module M of the software is trusted, means the behavior of the module M is consistent with the trusted behavior intended policy of the DTAC. Trusted behavior intended policy consists two parts: the intended behavior of the module M, and the jumping relations of the module M and the forward and backward modules.

In this paper, the trusted behavior intended policy is to mean the software's intended behavior describing sets or the Software Behavior Authentication Code.

As is presented before, the software behavior's trusted state is influenced by the module behavior. In the paper, we use the transfer function $f(k_i, M_i)=k_{i+1}$ to express the module software M_i makes the software trusted state transfer from k_i to k_{i+1}. The trusted module behavior will not change the creditability of the software behavior, based on which we make the definition 3.

Definition 3: If the software behavior state k_i and the module behavior M_i is trusted, meanwhile $f(k_i, M_i)= k_{i+1}$, then the software behavior state k_{i+1} is trusted.

According to the definition 3, we can conclude the theorem 1.

Theorem 1: At a given time t, if the software state k_t is trusted, and every software behavior module M_i ($1 \leqslant i \leqslant n$) of the software module behavior sequence $M=M_1M_2 \cdots M_n$ is trusted, then the software state $k_{t+1}(=f(k_t,M))$ is trusted, after the occurrence of the module behavior M.

The significance of the theorem 1 is quite important. Firstly, the state space of the software is infinite, which makes it hard to distinguish the trustiness of the software state. But, on the base of the theorem 1, at a given time, we can verify the trustiness of the software state, and the afterward module behavior to solve the problem of judging any software states' trustiness. Secondly, theorem 1 helps to transform the measurement and verification of the software behavior state to the module behavior, which provide the practical process of software behavior trustiness authentication with well theory basis.

It is known, according to the analysis, that the software behavior state's authenticating process can be described by the determined finite state automata (DFA). For software in the process of execution, it contains an initial state and one or more terminal states. We regard the trusted module set as the input alphabet set (conversion condition), and every trusted module's input will change the current state to another new state. On the basis of the theorem 1, we can represent the software behavior trustiness authentication framework using DFA as follows:

The according five tuples of the DFA is $M=(K,\Sigma,f,S,Z)$:

1) K is a finite set containing multiple kinds of the software's trusted state; each element in K is a trusted state.
2) Σ is a finite input alphabet set; each element in it is regarded as an input symbol (it is the module behavior M_i in this paper).
3) f is the transfer function, which is the mapping relation of the $K\times\Sigma\rightarrow K$. For example, if $f(k_i, M_i)=k_j$, ($k_i\in K$, $k_j\in K$), it means that the current trusted state is k_i, when the input symbol is the API function invoking sequence module M_i, the state of the software transfers to the next state k_j, and we consider the k_j as the subsequent trusted state of k_i.
4) $S\in K$ is the only one initial state. In this paper, it means the software's initial trusted state, which is unique.
5) Z is the set of the terminal states, which indicates the software's termination state and it can be more than one state.

DFA $M=(K,\Sigma,f,S,Z)$ and its behavior can expressed as follows:

```
K:= S;
M:=getModule;
While BeTrusted(M) do
  {
      K:=f(K,M);
      if K∈Z then return ('yes')
      M:= getModule;
  };
return ('no')
```

2.3 Implementation Method of Software Behavior' Dynamic Trustiness Authentication Framework

2.3.1 Process of the Software Dynamic Trustiness Authentication. We describe the software intended behavior based on the invoking sequences of the win32 API, so we define SIBDS and AM as below.

Definition 4: ***SIBDS***. SIBDS (Software Intended Behaviors Describing Sets) is an anticipated behavior set which describes the normal API invoking sequences and its interrelations. It is composed of API functions Module Set (AMS) and Modules Relationship Set (MRS), say SIBDS=<AMS, MRS>.

Definition 5: AM. AM (API functions invoking Module) refers to an API functions module which has a relative stable and independent sequence of API functions invoking when the software runs normally.

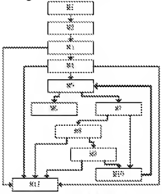


Figure 2. Call graph of AM of sample program

In this paper, we divide all the API invoking sequences into many independent API invoking sequence module according to the program flow and represent the relations and the content of all the modules using SIBDS. In this way, AMS and MRS can present the software behavior with the API function invoking sequence module in different levels. Figure 2 shows an example of our sample program (M_i is AM).

After obtaining AMS and MRS of the software, on the basis of the trustiness authentication theory introduced in section 2.2, we can build a corresponding finite state graph to describe the whole process of the software's trustiness authentication.

Figure 3 is the finite state graph built on the basis of representing the software's trustiness authentication process using determined finite state automata. It describes the software's trustiness authentication process from the level of API function invoking sequential module.

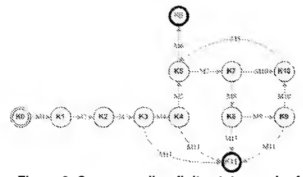


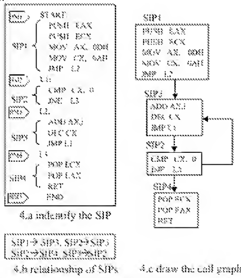
Figure 3. Corresponding finite state graph of Figure 2

Further more, we can define and describe each AM to describe the software behavior completely. With a moderate match algorithm, we can do the module behavior trustiness authentication to AM and form a complete authentication system and framework.

2.3.2 Extraction and Optimization of AM. When extracting the software intended behavior, we firstly do static disassemble to the software's binary executable.

According to the disassemble code, we can insert "Module BookMark"(BM) into the following places: (i) the start address of the current function;(ii)the address where the jump instructions jump to; (iii) the next instruction's address of the jump instructions; (iv) the address of return instruction.

After inserting some Module Bookmarks (BM) into the assembled code, we can divide the code into several SIPs (Sequential Instruction Pieces), and all of these SIPs form SIPS (Sequential Instruction Pieces Sets).



```
BM1  START
         PUSH  EAX
         PUSH  ECX
SIP1 {   MOV   AX, 0DH
         MOV   CX, 0AH
         JMP   L2
BM2  L1:
SIP2 {   CMP  CX, 0
         JNE   L3
BM3  L2:
         ADD AX,1
SIP3 {   DEC CX
         JMP L1
BM4  L3
         POP ECX
SIP4 {   POP EAX
         RET
BM5      END
```

4.a indentify the SIP

```
SIP1→SIP3, SIP2→SIP3
SIP2→SIP4, SIP3→SIP2
```

4.b relationship of SIPs

```
SIP1
PUSH  EAX
PUSH  ECX
MOV   AX,  0DH
MOV   CX,  0AH
JMP   L2

SIP3
ADD AX,1
DEC CX
JMP L1

SIP2
CMP  CX, 0
JNE  L3

SIP4
POP ECX
POP EAX
RET
```

4.c draw the call graph

Figure 4. Process of dividing assemble code into SIPS

Then, we give each SIP an unique label to identify the SIP and record the jumping relationship between different SIP, say SIPRS (Sequential Instruction Pieces Relationship Sets), using directed edge (each directed edge represents a jumping relation in the call graph), draw the flow graph according to the SIPS and SIPRS and finally we have a call graph of a AM. In the same way, we can draw other AM's call graph accordingly. The drawing process is shown below in Figure 4.

In the process of extracting the sequence of API invoking, we focus on the functions (self defined functions and API functions) invoking instructions and all kinds of jumping instructions (e.g., JMP, JXX, LOOP, RET and so on) and neglecting all the other instructions. In this way, the graph we finally get is an API invoking relationship graph. Another extraction method is proposed in the literature. [12]

2.3.3 Storage Structure of the API function invoking module (AM). MRS describes the trusted relations between AMs at a high level. To ensure that the software is trusted at the level of API function, it is necessary to ensure the trustiness of each AM.

In the MRS's FSM (Finite State Machine), AM is the input element of the finite input alphabet. At the time of the state conversion of the FSM, the trustiness authentication of the input element is quite essential, which makes it necessary to define an efficient storage structure and provide a complete authentication method of AM.

Based on software's API function invoking flow graph, we can make the definition of the AMS.

The hierarchical relationship among the data structures of SIBDS is described as follows:

```
|— SIBDS (Software Intended Behaviors Describing Sets)
  |— AMRS (API Modules Relationship Set)
  |— AMS (API Module Set)
    |— AM (API functions invoking module)
      |—SIPRS (SIP Relationship Sets)
      |— SIPS (Sequential Instruction Pieces Sets)
        |— SIP (Sequential Instruction Pieces)
          |— APIFC (API Functions Calling)
          |—SDFC (Self-Defined Functions Calling)
```

Limited to the page cover, we won't give the related storage data structure in detail in this paper.

2.4 Software Behavior Monitoring Method and Software Trustiness Authentication Algorithm

In this paper, we monitor the real execution process of the software to get a real-time API functions invoking sequence, comparing which with the Software Behavior Authentication Code to realize the trustiness authentication and control of the software behavior.

The software behavior authentication algorithm, in essence, is a matching algorithm which matches the real API functions invoking sequence with the Software Behavior Authentication Code (storage definition of the SIBDS).

3. Related Experimental Data and Result Evaluation

Overflowing attack and malicious remote thread injection are two typical attack techniques to deviate the software's dynamic behavior. Overflowing attack is the main skills used by hackers to conduct the remote attack, and remote thread injection is a common way used by local host's vicious code to hide itself from being detected. To test the dynamic trustiness authentication's validity of the framework we proposed in this paper, we carry out three typical experiments: the first one is the dynamic trustiness authentication experiment in the safe environment, it mainly examines whether the framework can authenticate the normal process behavior or not; the other two test is aiming at the two typical dynamic intrusion methods to test the framework's detecting ability of the malicious attack.

The exact number of AM (API function invoking Module) and SIP (Sequential Instruction Piece) in our sample program is shown as Table 1:

Table 1. Number of AM and SIP

AM	Self-defined function calling module	SIP
11	13	110

3.1 Software Dynamic Trustiness Authentication Experiment in Safety Environment

Normal API function invoking sequence from the main thread of the sample program is shown as below:

Table 2. Normal API Function invoking sequence

......
GetSystemDirectoryA->GetModuleFileNameA->lstrcat->
CopyFileA->RegOpenKeyExA->RegSetValueExA->
RegCloseKey->GetProcAddress->FreeLibrary->
WSAStartup->socket->htons->bind->listen->accept->
GetSystemMetrics->GetSystemMetrics->send->recv->
Recv->sprintf->strcpy->MessageBoxA->closesocket->
accept......

The API function invoking sequence accord with AM and MRs, which is described as Table 3.

The API function invoking in italic is the overflowing point. The API "recv" receives data from another network terminal, "sprintf" does the string formatting work , "strcpy" is used to copy the string and "MessageBoxA" is used to show the string data.

In normal sense, the monitor system can record the invoking sequence completely in real-time and match it with SBAC. The experiment result indicates that the software can pass dynamic trustiness authentication when it follows the normal sequence.

Table 3. API function invoking sequence and AMs

API Function Invoking Sequence	Corresponding AM
GetSystemDirectoryA-> GetModuleFileNameA->lstrcat-> CopyFileA->RegOpenKeyExA-> RegSetValueExA->RegCloseKey->	AM1
GetProcAddress->FreeLibrary->	AM2
WSAStartup->	AM3
socket->htons->bind->listen->	AM4
accept->GetSystemMetrics-> GetSystemMetrics->	AM5
send->recv->recv->sprintf->strcpy-> MessageBoxA->	AM7
closesocket->	AM10
accept......	AM5 and other AMs

3.2 Buffer Overflowing Detecting Experiment

We conduct an overflowing attack and record the corresponding API invoking sequence. The real API function invoking sequence when suffering the overflowing attack is shows as Table 4.

It can be found that when the overflowing attack occurs, the system monitor is able to report that there happens a lots other API function invoking after the calling of "MessageBoxA", and these unexpected APIs are indeed called by the shellcode, which can't be trusted by DTAC.

Table 4. API function invoking sequence when suffering the overflowing attack

......
GetSystemDirectoryA->GetModuleFileNameA->lstrcat->
......
Recv->sprintf->strcpy->MessageBoxA->LoadLibraryA->
Sleep->WaitForSingleObjectEx->ioctlsocket->recv->
CreateProcessA->TerminateProcess->closesocket->
accept......

3.3 Remote Unknown Thread Injection Detecting Experiment

The well-known Trojan program named gray-pigeon injects itself as a remote thread into a specified process. In this experiment, we configure the gray-pigeon server program to make it inject into a process which is monitored under the dynamic authentication system. The experiment result shows that the new thread is not able to pass the dynamic trustiness authentication.

The experiment results and comparison, which can demonstrate the efficacy of the dynamic trustiness authentication framework, are shown in Table 5.

Table 5. Experiments result and comparison

Intended AMs	Safety Environment		Buffer Overflowing		Remote Thread	
	①	②	①	②	①	②
AM1	√	√	√	√	×	\
AM2	√	√	√	√	\	\
AM3	√	√	√	√	\	\
AM4	√	√	√	√	\	\
AM5	√	√	√	√	\	\
AM7	√	√	×	\	\	\
AM10	√	√	\	\	\	\
AM5	√	√	\	\	\	\
…	√	√	\	\	\	\

① : Is API Function invoking sequence part according with the trusted behavior of AM? ("•" means "Yes", "x" means "No", " \ " means that the behavior of software is not trusted)

②: Is the order of AMs according with MRS?

4. Conclusion

In this paper, we propose a dynamic trustiness authentication framework based on the integrity of the software behavior. This framework carries on the analysis to the binary procedure to obtain the Software Intended Behaviors Describing Sets. Through estimating the API function call sequence which is executed by real-time monitoring process whether to conform to the software anticipated behavior, the framework can examine the integrity of the software behavior, thus authenticate the dynamic trustiness of the software.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (Nos: 60673071, 60743003, 90718005, 90718006) and the National High Technology Research and Development Program of China (863 Foundation) (Nos: 2006AA01Z442, 2007AA01Z411)

Reference

[1] Shen Changxiang, Zhang Huanguo, Feng Dengguo ,*et al*, "Summary of Information Security", Science in China Press, 2007(2), pp. 129-150. (Ch)

[2] Lin Chuang, Peng Xuehai, "Research on Trustworthy Networks", Chinese Journal of Computers, 2005(5), pp. 751-758. (Ch)

[3] Duan Yunsu, Liu Xin, Chen Zhong, "Analysis on Combined Security Efficiency and Vulnerability for Information System Security Evaluation", Journal of Peking University(Natural Science Edition), 2005(3), pp. 484-490.(Ch).

[4] Zhang Huanguo, Luo Jie, Jin Gang, "Development of Trusted Computing Research", Journal of Wuhan University(Natural Science Edition), 2006(5) , pp. 513-518. (Ch)

[5] Forrest S, Hofmeyr S, Somayaji A, *et al.* "A Sense of Self for Unix Processes", *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. Washington, D C: IEEE Computer Society Press, 1996, pp. 120-128.

[6] Hofmeyr S, Forrest S, Somayaji, "A. Intrusion Detection Using Sequences of System Calls", *Journal of Computer Security*, 1998 (6), pp. 151-180.

[7] Wepsi A., Dacier M. , Debar H. , "Intrusion detection using variable length audit trail patterns", Proceedings of t he 3rd International Workshop on Recent Advances in Intrusion Detection , Springer2Verlag , London , U K, 2000 , pp.110～129.

[8] Sekar R. , Bendre M. , Dhurjati D. , *et al*, "A fast automation-based method for detecting anomalous program behaviors", Proceedings of t he 2001 IEEE Symposium on Security and Privacy. IEEE Computer Society , Washington , DC ,USA , 2001 , pp. 144～149.

[9] Xu Jianyun, Sung Andrew H, Chavez Patrick, *et al*, "Polymorphic Malicious Executable Scanner by API Sequence Analysis", *Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*. Washington D C: IEEE Computer Society Press, 2004, pp. 378-383.

[10] Christodorescu Mihai, Jha Somesh, "Static Analysis of Executables to Detect Malicious Patterns", *Proceedings of the 12th USENIX Security Symposium*, Washington D C. Berkeley, pp.169-186.

[11] Bergeron J, Debbabi M, Desharnais J. "Static Detection of Malicious Code in Executable Programs", *Symposium on Requirements Engineering for Information Security*. http://www.sreis.org/old/2001/papers/sreis014.pdf.

[12] Su Purui, Yang Yi, "Intrusion Detection Model Based on Executable Static Analysis", Chinese Academy of Sciences. 2006(9), pp. 1572-1578.(Ch)